Numerical Method for Finding Minimum Distance to an Ellipsoid

Stanley Chan
h5chan@ucsd.edu

The goal is to solve this equality constrained problem:

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & \|y-x\|^2 \\ \text{subject to} & x^T A x = 1. \end{array} \tag{1}$$

We form the Lagrangian,

$$L = \|y-x\|^2 + \lambda(1 - x^T A x) \tag{2}$$

Take derivatives on both sides,

$$\begin{aligned} \nabla_x L &= (y-x) - \lambda(Ax) \\ \nabla_\lambda L &= x^T A x - 1 \end{aligned}$$

Necessary condition for an optimal point is $\nabla_x L = 0$ and $\nabla_\lambda L = 0$. Thus we have

$$(y-x) - \lambda(Ax) = 0 \tag{3}$$

$$x^T A x = 1 \tag{4}$$

For the first equation, we have

$$\begin{aligned} y &= (I + \lambda A)x \\ \Rightarrow x &= (I + \lambda A)^{-1} y \end{aligned}$$

Put this into Equation (4), we have

$$y^T (I + \lambda A)^{-T} A (I + \lambda A)^{-1} y = 1 \tag{5}$$

Now note that since A is symmetric (assume non singular), we can decompose A into

$$A = V^T D V, \tag{6}$$

where columns of V form eigenvectors, and D is a diagonal matrix with associated eigenvalues. Remark: we can always scale V so that $V^T V = I$ and $V V^T = I$.

With these we have

$$\begin{aligned} (I + \lambda A)^{-1} &= (V^T V + \lambda V^T D V)^{-1} \\ &= \left[V^T (I + \lambda D) V\right]^{-1} \\ &= V^{-1} (I + \lambda D)^{-1} V^{-T} \\ &= V^T (I + \lambda D)^{-1} V. \end{aligned}$$

Thus

$$\begin{aligned} (I + \lambda A)^{-T} A (I + \lambda A)^{-1} &= \left[V^T (I + \lambda D)^{-1} V\right] (V^T D V) \left[V^T (I + \lambda D)^{-1} V\right] \\ &= V^T (I + \lambda D)^{-1} D (I + \lambda D)^{-1} V \end{aligned}$$

Put this into Equation (5) we have

$$y^T V^T (I + \lambda D)^{-1} D (I + \lambda D)^{-1} V y = 1.$$

The middle part of the matrix equality can be simplified as

$$M = (I+\lambda D)^{-1} D (I+\lambda D)^{-1} = \begin{pmatrix} \frac{d_1}{(1+\lambda d_1)^2} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{d_n}{(1+\lambda d_n)^2} \end{pmatrix}$$

So the matrix equality becomes

$$y^T V^T M V y = 1. \tag{7}$$

Let $\mathbf{y} = [y_1 \dots y_n]^T$, $V = [\mathbf{v}_1 \dots \mathbf{v}_n]$, $M = \text{diag}\left\{\frac{d_1}{(1+\lambda d_1)^2}, \dots, \frac{d_n}{(1+\lambda d_n)^2}\right\}$. Then left hand side of Equation (7) can be written as

$$\mathbf{y}^T V^T M V \mathbf{y} = \left[\sum_{i=1}^{n} y_i \mathbf{v}_i\right]^T \begin{pmatrix} \frac{d_1}{(1+\lambda d_1)^2} & & \\ & \ddots & \\ & & \frac{d_n}{(1+\lambda d_n)^2} \end{pmatrix} \left[\sum_{i=1}^{n} y_i \mathbf{v}_i\right]$$

Therefore, if we let $\mathbf{q} = \sum_{i=1}^{n} y_i \mathbf{v}_i$, and denote components of $\mathbf{q}$ by q_k, then the above equation is equivalent to

$$\sum_{k=1}^{n} \frac{d_k q_k^2}{(1+\lambda d_k)^2} = 1, \tag{8}$$

which does not seem to have closed form solutions.

**Analytic solution is available only for special case (circle) where $d_i = d$:

$$\begin{aligned} & \mathbf{y}^T V^T \begin{pmatrix} \frac{d}{(1+\lambda d)^2} & & \\ & \ddots & \\ & & \frac{d}{(1+\lambda d)^2} \end{pmatrix} V \mathbf{y} = 1 \\ \Rightarrow \quad & \frac{d}{(1+\lambda d)^2} \mathbf{y}^T V^T V \mathbf{y} = 1 \\ \Rightarrow \quad & \frac{d \|\mathbf{y}\|^2}{(1+\lambda d)^2} = 1 \end{aligned}$$

Thus

$$\lambda = \frac{\sqrt{d \|\mathbf{y}\|^2} - 1}{d}. \tag{9}$$

Secant Method
We now discuss a fast numerical implementation for solving Equation (8).
Let
$$F(\lambda) = \sum_{k=1}^{n} \frac{d_k q_k^2}{(1+\lambda d_k)^2} - 1. \tag{10}$$
Our goal is to find λ such that $F(\lambda) = 0$.

We propose using secant method, where the update is done as
$$\lambda_{k+1} = \lambda_k - F(\lambda_k)\frac{\lambda_k - \lambda_{k-1}}{F(\lambda_k) - F(\lambda_{k-1})}.$$

Why not Newton?
Secant method is a modification of Newton's method. Compared to Newton, Secant is more numerically stable. In Newton's method, the update is performed as
$$\lambda_{k+1} = \lambda_k - \frac{F(\lambda_k)}{F'(\lambda_k)}.$$
However, $F(\lambda)$ is a monotonic increasing function (because $\lambda > 0$, $d_i > 0$), and it flattens when λ increases. In other words, the derivative is almost zero when λ is large. If initial guess is not well chosen to make $F'(\lambda)$ small, then very soon Newton will diverge.

However for secant, since we perform the derivative as a finite difference, the stability is significantly improved. As long as λ_{k+1} and λ_k are chosen such that $|F(\lambda_{k+1}) - F(\lambda_k)|$ is within reasonable numerical range, then secant converges. Such condition can be satisfied more easier than for Newton.

Why not integration?
We can integrate Equation (10) so that the problem is transferred to
$$\text{minimize} \quad G(\lambda) = \sum_{k=1}^{n} \frac{d_k q_k^2}{(1+\lambda d_k)} + \lambda. \tag{11}$$
Then we can use Newton's method to determine the minimum point, as this problem is convex. However, this is not necessary, because in solving the minimization problem, what we are doing is really finding a stationary point. For convex problem, how do we efficiently find such stationary point? We solve $\nabla G = 0$ by Newton. But then it goes back to Equation (10). To present the idea in a more precise way, observe that the update for minimizing Equation (11) is
$$\lambda_{k+1} = \lambda_k - \frac{G'(\lambda_k)}{G''(\lambda_k)}.$$
But since $G' = F$, this is exactly the same as finding a zero in Equation (10) by Newton method.

Implementation
Detailed MATLAB implementation and numerical results are given in Appendix A.
A few key notes:

1. Speed: If we only compare secant method v.s. fzero, then in a typical 100 dimension problem secant takes 0.003825 seconds whereas fzero takes 0.006458 seconds.
2. Interval Search: In case where the function is non-smooth, (if you're so unlucky), then in order to give a good initial point for secant we need a bisection (around 5 iterations).
3. Robustness: Accurate results for most situation. Fails to converge when function is extremely non-smooth (but in this case fzero also gives a wrong answer).
4. Solutions: The program outputs the minimum distance $\|x - y\|$ and Lagrange multiplier λ.

Appendix A MATLAB implementation for min distance to ellipse

Main Program

```
%% Main Program

clear all
close all

global q d A options

%% Setup
load 'parameters';   % parameters include D, V
y       = 10*(rand(100,1));
d       = diag(D);
q       = V*y;

options.display         = true;
options.interval_search = true;

t0               = cputime;
[dist, lambda]   = min_elp_dist(y);
itn_time         = cputime - t0;

fprintf('itn_time   lambda       dist \n');
fprintf('%3f   %3f    %3f \n', itn_time, lambda, dist);
```

minimum distance from point to ellipse

```
function [dist, lambda] = min_elp_dist(y, A, options)
% [dist, lambda] = min_elp_dist(y)
% computes the minimum distance from a point y to an ellipsoid x'Ax<= 1 by
% solving
%
%        minimize    ||x - y||
%        subject to  x' A x <= 1.
%
% Input:    y      -    a vector
% Output:   dist   -    distance to the ellipse
%           lambda -    the associated lagrange multiplier
%
% Global:   A      -    Ellipse parameter
%           D      -    Diagonal matrix storing eigenvalues of A
%                       A = V' D V;
%           d      -    eigenvalues stored in a vector (same as D)
%           q      -    premultiplied vector V*y, where V is the
%           eigenvector of A
%
% option:   display         -   Set 'true' if display iteration
%           interval_search -   Set 'true' if bisection interval estimation is
%           needed. This option is not recommended unless the point y is
%           likely to cause singularity.
%
% Algorithm: The algorithm is based on bisection interval search and secant
% method. Interval search provides a good initial guess. Secant search find
% a zeros based on first derivative approximation. [1]
%
```

```
% [1] David Kincaid, Ward Cheney, "Numerical Anlaysis", 3rd Edition.
%
% Stanley Chan
% Department of Electrical and Computer Engineering
% UC San Diego
%
% 8 Mar, 2008

global q d

max_itn = 50;
Infty   = 1e16;

[V D]   = eig(A);
V       = V';
d       = diag(D);
q       = V*y;

d_mean  = mean(d);
y_norm  = norm(y);
x0      = (sqrt(d_mean*y_norm^2)-1)/d_mean;

if options.interval_search
%% Interval Search
%  Bisection for *Rough* interval estimation
a       = 0;
b       = 2*x0;
c       = a + (b-a)/2;
Fa      = myfun(a);
Fc      = myfun(c);
Fb      = myfun(b);

itn     = 1;
TOL     = 1e2;
if options.display
    fprintf('Searching acceptable interval: \n');
    fprintf('itn     a         c        b \n');
end
while ( itn < max_itn )&&(( abs(Fa - Fc) > TOL )||( abs(Fb - Fc) > TOL ))
    c       = a + (b-a)/2;
    Fa      = myfun(a);
    Fc      = myfun(c);
    Fb      = myfun(b);
    if sign(Fa)~=sign(Fc)
        b = c;
    else
        a = c;
    end
    if options.display
        fprintf('%3d %3f %3f %3f \n', itn, a, c, b);
    end
    itn = itn + 1;
end
if options.display
```

```
    fprintf('Interval found! \n');
    fprintf('Interval = [%3f  %3f   %3f]\n\n', a, c, b);
end
else
    c = x0;
end

%% Secant Method
%  Secant search for zeros
x       = [c-10*sqrt(eps) c+10*sqrt(eps)];
err     = Infty;
k       = 2;
TOL     = 1e-5;


if options.display
    fprintf('Searching lambda within interval: \n');
    fprintf('itn       x          err\n');
end
tic
while ( k < max_itn )&&( err > TOL )
    deltaF      = (myfun(x(k))-myfun(x(k-1)));

    if abs(deltaF)<=1e-10 % Stop if derivative too small!
        error('Derivative almost zero! Impossible for Secant. Try fzero.');
    else
        x(k+1)  = x(k) - myfun(x(k))*(x(k)-x(k-1))/(deltaF); % Secant update

        err     = abs(x(k) - x(k-1)).^2;
        if options.display
            fprintf('%3d %3d %3d \n', k, x(k), err);
        end
    end

    k = k + 1;
end
toc

lambda = x(k);

if options.display
    fprintf('Lambda found! \n');
    fprintf('Lambda = %3f \n\n', lambda);
end

%% Find distance
z        = (eye(size(A)) + lambda*A)\y;
dist     = norm(z-y);

%% myfun.m
function F = myfun(x)
global q d
F = -sum((q.*q).*(d./((1+d*x).^2)))+1;
```

Numerical Results

```
Searching acceptable interval:
itn     a         c        b
  1 0.000000 13.144884 13.144884
  2 6.572442 6.572442 13.144884
  3 6.572442 9.858663 9.858663
Interval found!
Interval = [6.572442  9.858663   9.858663]

Searching lambda within interval:
itn       x          err
  2 1.085866e+001   1
  3 9.716151e+000 1.305334e+000
  4 9.740574e+000 5.964677e-004
Lambda found!
Lambda = 9.737018

itn_time   lambda       dist
0.109201   9.737018    62.366681
```

Appendix B Experiment with `fzero.m`

We now show some numerical experiment results. Equation (8) can be solved using standard root finding routine `fzero.m` in MATLAB. `fzero.m` runs two parts: (i) searches an interval where there is sign change; (ii) interpolate the solution within the interval. It is a combination of bisection, secant, and inverse quadratic interpolation methods. (c.f. MATLAB `help fzero`)
As an numerical example, we have the following parameters

write parameters

```
%% Write Parameters
A     = round(10*rand(100,100));
A     = (A + A')/2; % so the eigenvalues are real

[V D] = eig(A);
V     = V';         % to fit our notation
D     = abs(D);     % so that it's a real ellipse

save('parameters', 'V', 'D');
```

The results are shown in the next page. Running on a Intel Core 2 Duo 2.33 GHz, 2GB RAM, Win Vista O/S, the iteration takes 0.004703 seconds to complete. Initial guess was set as the solution to the special case (Equation (9)), with $d = \sum d_k/n$.

numerical example

```
function Trial_2
global V D y

% clear all
% close all

%% Setup
load 'parameters';   % parameters include D, V
y     = 20*(rand(100,1));

%% initial guess
d_mean = mean(diag(D));
y_norm = norm(y);
x0     = (sqrt(d_mean*y_norm^2)-1)/d_mean;

%% Main
tic
options = optimset('Display','off');
x = fzero(@myfun,x0,options)
toc

%% Function Evaluation
function F = myfun(x)
global V D y
M = D./((1+x*D).^2);
F = y'*V'*M*V*y-1;
```

Numerical Results

```
Search for an interval around 24.0802 containing a sign change:
 Func-count    a          f(a)             b          f(b)        Procedure
    1         24.0802    -0.0197274       24.0802    -0.0197274   initial interval
    2         23.3991     0.0365158       24.0802    -0.0197274   search

Search for a zero in the interval [23.3991, 24.0802]:
 Func-count    x          f(x)             Procedure
    2         24.0802    -0.0197274        initial
    3         23.8413  -0.000533115        interpolation
    4         23.8347  2.36915e-007        interpolation
    5         23.8347 -9.48106e-011        interpolation
    6         23.8347 -1.11022e-016        interpolation
    7         23.8347 -1.11022e-016        interpolation

Zero found in the interval [23.3991, 24.0802]

x =

   23.8347
```